AIAA 2000-0808

A Domain-Decomposed Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries

M. J. Aftosmis
Mail Stop T27B
NASA Ames Research Center
Moffett Field, CA 94404

M. J. Berger and **G. Adomavicius**
Courant Institute
12 Mercer St.
New York, NY 10012

38th Aerospace Sciences Meeting and Exhibit
10-13 January 2000 / Reno NV

A Domain-Decomposed Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries

M. J. Aftosmis[†]
Mail Stop T27B
NASA Ames Research Center
Moffett Field, CA 94404
aftosmis@nas.nasa.gov

M. J. Berger[‡] and **G. Adomavicius**[¥]
Courant Institute
12 Mercer St.
New York, NY 10012
berger@cims.nyu.edu

Preliminary verification and validation of an efficient Euler solver for adaptively refined Cartesian meshes with embedded boundaries is presented. The parallel, multilevel method makes use of a new on-the-fly parallel domain decomposition strategy based upon the use of space-filling curves, and automatically generates a sequence of coarse meshes for processing by the multigrid smoother. The coarse mesh generation algorithm produces grids which completely cover the computational domain at every level in the mesh hierarchy. A series of examples on realistically complex three-dimensional configurations demonstrate that this new coarsening algorithm reliably achieves mesh coarsening ratios in excess of 7 on adaptively refined meshes. Numerical investigations of the scheme's local truncation error demonstrate an achieved order of accuracy between 1.82 and 1.88. Convergence results for the multigrid scheme are presented for both subsonic and transonic test cases and demonstrate W-cycle multigrid convergence rates between 0.84 and 0.94. Preliminary parallel scalability tests on both simple wing and complex complete aircraft geometries shows a computational speedup of 52 on 64 processors using the run-time mesh partitioner.

1 Introduction

THIS paper documents the development of a new inviscid flow solver designed for Cartesian meshes with embedded boundaries. The ability of these meshes to automatically handle extremely complicated geometries has led to a substantial increase in their use over the past decade[1]–[8]. As recently as five to ten years ago, mesh generation was frequently the most time consuming task in a typical CFD cycle. Adaptive Cartesian mesh generation methods are capable of producing millions of cells around complex geometries in minutes and have substantially removed this bottleneck.

Why write yet another Euler solver? With robust mesh generation largely in-hand, solution time resurfaces as the pacing item in the CFD cycle. The current work attacks this issue by designing a scalable, accurate Cartesian solver with robust multigrid convergence acceleration. Our primary motivation is to gain efficiency by capitalizing on the simplifications and specialized data structures available on Cartesian grids. Significant savings in both CPU time and storage may be realized by taking advantage of the fact that cell faces are coordinate aligned. In addition, second-order methods with good limiters are generally easier to design and perform more robustly on uniform Cartesian meshes.

Secondly, in any embedded-boundary Cartesian solver, the body-intersecting *cut-cells* demand special attention. These cells can impose a substantial burden on the numerical discretization since the arbitrary nature of geometric intersection implies that one cut-cell may be orders of magnitude smaller than its neighboring cells. This fact contrasts sharply with the comparatively smooth meshes that are generally found on a good quality structured or unstructured mesh. Substantial research into these cut-cell issues have been studied by references [9],[10],[6],[8], and [12] (among others) and we hope to take advantage of on this investment.

Thirdly, this work investigates a multigrid strategy that is specialized for adaptively refined Cartesian meshes. In our approach, all grids in the multigrid hierarchy cover the entire domain and include cells at many refinement levels. The smoother therefore iterates over the entire domain when it is invoked on any grid in the hierarchy. In this respect, the approach shares more with agglomeration or algebraic multigrid techniques than with many other Cartesian or AMR methods which iterate over only cells at the same level of refinement[13], or employ a "lifting" strategy[14]. The decision to treat all grids globally follows in large part from considerations of parallel efficiency.

A major unanswered question about multigrid for non-body-fitted Cartesian meshes concerns the degree to which geometric fidelity must be preserved on coarser grids in the hierarchy to avoid adversely impacting the convergence rate. With very little overhead, our design permits a variable level of geometric detail to be retained on the coarser grids so that this issue may be investigated.

Finally, a truly scalable solver demands that parallel considerations be incorporated into its architecture from the beginning. The current work adopts a domain decomposition approach with explicit message passing. At run time, space-filling curves are generated on-the-fly and are used to partition the mesh for any number of subdomains (and proces-

---

† Research Scientist, Senior Member AIAA
‡ Professor, Member AIAA
¥ Research Assistant

sors). Although the solver explicitly passes messages between subdomains, the implementation uses a shared-memory approach (currently aimed at distributed shared memory architectures). This approach provides direct control of memory placement and CPU scheduling which is necessary to reliably obtain good scalability. These issues differentiate this approach from more routine implementations.

This paper documents the current status in the development of this solver. Section 2 presents an overview of the spatial discretization and includes a mesh refinement study to provide initial verification and accuracy assessment. Section 3 motivates and describes a new multigrid coarsening strategy for adapted Cartesian meshes. Examples in this section demonstrate preliminary multigrid convergence rates on domains with embedded boundaries in both subsonic and transonic flow regimes. Section 4 discusses the parallel domain decomposition, and is followed by scalability results on both a simple wing and a full aircraft geometry.

2 Governing Equations and Discretization

The three dimensional Euler equations governing inviscid flow of a perfect gas may be written for a control volume Ω with closed boundary $\partial\Omega$ as:

$$\iiint_{\Omega} \frac{\partial}{\partial t} Q dV = -\oint_{\partial\Omega} (\bar{F} \cdot \hat{n}) dS \qquad (1)$$

where Q is the state vector of conserved variables $Q = [\rho, \rho u, \rho v, \rho w, \rho E]^T$ and $\hat{n}$ is an outward facing unit vector. $\bar{F}$ is the tensor of flux density with components for all three Cartesian fluxes. The mean-preserving, cell-centered implementation locates the state vector at the centroid of all control volumes as shown in Figure 1a. Assuming control volumes with planar faces which are fixed in time, and applying a midpoint quadrature rule, we arrive at the semi-discrete form of the governing equations,

$$\frac{d}{dt}\tilde{Q} = -\frac{1}{V_\Omega} \sum_{faces} \bar{F} \cdot \hat{n} S_j \qquad (2)$$

where j is a running variable over the faces of the control volume and $\tilde{Q}$ is now the integral cell average of the state.

2.1 Spatial Discretization

A unique value of the numerical flux on each face, $\bar{F}_j$ is computed using either the approximate Riemann solver of ref. [11] or van Leer's flux vector splitting, using the implementation of ref.[15].

To produce a second-order accurate scheme, the Riemann solver on each face uses linearly reconstructed data from the left and right neighboring cells. Both flux functions have been implemented using either primitive or conserved variables in the reconstructed state. A least-squares procedure provides estimates of the gradients within each cell using data from the distance-one face neighbors in the connectivity graph as shown in Figure 1b. The connectivity matrix of each cell is inverted using the normal equations to compute the gradients within each control volume. Options exist to apply either Minmod or Venkatakrishnan's flux limiter[16] to control overshoots in the reconstructed data.

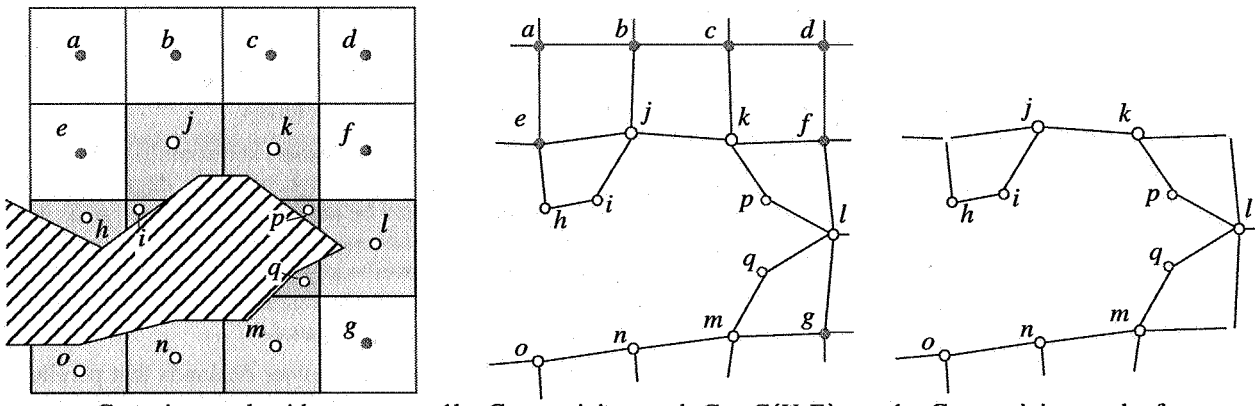
2.2 Data Structures

A typical swatch of Cartesian mesh is shown in the left frame of Figure 1. It is instructive to examine the connectivity graph, $G = \{V, E\}$, as sketched in Figure 1b. In this dual view of the mesh, flow-through faces in the physical mesh form the set of edges, E, and the control volumes, V, appear as nodes.

Using notation similar to that taken by ref. [6], G can be decomposed into two parts.

$$G = G_c + G_\perp \qquad (3)$$

$G_\perp = \{V_\perp, E_\perp\}$ is formed by the set of control volumes in V which are *uncut Cartesian cells*. The edges, $E_\perp$ in this subgraph, include only those edges in $\{E\}$ which connect two uncut Cartesian cells. The *cut graph*, $G_c = \{V_c, E_c\}$, consists of all control volumes, $V_c = V - V_\perp$, which exist within the set of Cartesian hexahedra which actually intersect the geometry, and any edge, $E_c = E - E_\perp$, which is incident upon at least one



1a. Cartesian mesh with embedded geometry

1b. Connectivity graph $G = G\{V, E\}$

1c. Connectivity graph of non-Cartesian cells and faces

Figure 1: Control volumes in a Cartesian mesh with embedded geometry showing *uncut Cartesian cells*, *a–g*, *cut-cells*, *h–o*, and *split-cells*, *p* and *q*. The connectivity graph, G may be divided into subsets $G_\perp$ and G_c where G_c includes all non-Cartesian control volumes and any edge in the connectivity graph connected to at least one non-Cartesian cell.

member of $\{V_c\}$. Its important to note that due to the presence of *split-cells* (like p and q in Figure 1a) the number of cut-control volumes does not have to be equal to the number of cut Cartesian hexahedra. The composition of the cut graph differs from that in ref. [6] since G_c includes the graph edges which connect cut-cells to uncut cells. Figure 1c further illustrates the composition of G_c.

The utility of this decomposition becomes clear when planning data structures for the control volumes and edges in the Cartesian mesh. The uncut graph includes only Cartesian cells and faces. In three dimensions with N^3 control volumes in the mesh, $G_\perp$ will contain $O(N^3)$ control volumes and faces. Since the number of cut hexahedra scales with the surface area of the geometry being meshed, G_c will contain only $O(N^2)$ cells. Geometric information for the Cartesian control volumes and faces in $G_\perp$ can be computed on-the-fly using the packed storage scheme of ref. [2] while more conventional unstructured data structures are used for recording the cut graph, G_c. Since the number of cut-cells becomes small with respect to the total number of cells on large meshes, the storage requirements for all cell size and position data asymptotically approaches the limit of 1.5 (64-bit) words presented in ref. [2].

In addition to less expensive data storage, specialization of the integration scheme within $G_\perp$ permits simplification of the operators evaluated on the right-hand-side of eq. (2). For example, the vector reconstruction operator which takes data from the cell centroid to the face centroids becomes a single multiply-add instruction for each element in the state vector. Similar simplifications are used in construction of the gradients within Cartesian control volumes. Timing results indicate that, on average, the residual computation in a pure Cartesian cell in $G_\perp$ requires approximately 1/3 the CPU time of a cut-cell in G_c.

Both the cut and uncut graphs are stored using *flat* data structures similar to those typically encountered in unstructured flow solvers. This approach contrasts with the hierarchical (tree-based) or logically rectangular storage schemes often encountered in the literature of Cartesian and AMR approaches. At the cost of an explicitly stored face-list, the unstructured approach facilitates reordering of the main arrays for both improved cache-performance and graph partitioning. It also easily supports anisotropic cell refinement as investigated in ref. [17].

2.3 Verification and Order of Accuracy

We use a model problem to both verify that our implementation correctly solves the Euler equations and to demonstrate the order of accuracy of the spatial discretization on actual meshes. This investigation relies upon a closed-form, analytic solution to the Euler equations for the supersonic vortex flow of ref. [20]. The presence of an exact solution permits examination of the truncation error of the discrete solution using a series of telescoping meshes. Since this is a shock-free flow, the measured order of accuracy is not corrupted by limiter action near shocks, and the behavior is indicative of the scheme's performance in smooth regions of a flow.

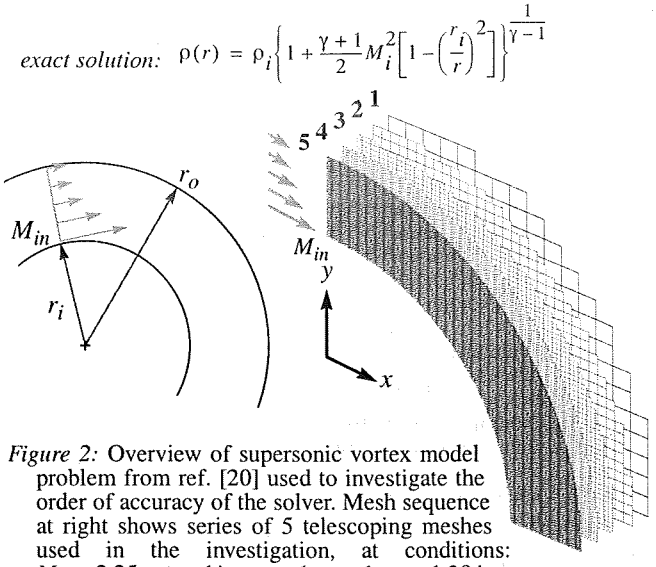


Figure 2: Overview of supersonic vortex model problem from ref. [20] used to investigate the order of accuracy of the solver. Mesh sequence at right shows series of 5 telescoping meshes used in the investigation, at conditions: $M_{in} = 2.25$, $p_{in} = 1/\gamma$, $\rho_{in} = 1$, $r_i = 1$, $r_o = 1.384$.

Although this example is only two dimensional, the full three dimensional solver was run using a 3-D geometry made by extrusion.

To investigate the local truncation error of the scheme, the domain was initialized with the exact solution and integrated one time step. The residual in each cell then offers a direct measure of the difference between the discrete scheme and the governing equations, including the effects of boundary conditions. Prior research on this topic suggests that the behavior of the global error mimics that of the one-step truncation error.[10][18][19]

Figure 2 presents an overview of the investigation. The sketch at the left shows the inviscid flow between two concentric circular arcs, while the frame at the right shows the sequence of five Cartesian meshes used in the investigation. The meshes were created by nested subdivision. The coarsest of these grids had 105 cells in a 2D slice, the finest had over 21000. All meshes in this example are uniformly refined.

Figure 3 contains a plot of the L1 norm of density error resulting from this analysis.

$$\|error\|_1 = \frac{\sum_{cells} V_j \left| \rho_{j_{exact}} - \rho_j \right|}{\sum_{cells} V_j \left| \rho_{j_{exact}} \right|} \tag{4}$$

The error plot is remarkably linear over the first 4 meshes, but shows signs of a slight tailing-off on the final mesh. Over the first 4 meshes, the average order of accuracy is 1.88. If the finest mesh is included, this estimate drops to a value of 1.82. Both of these slopes are comparable to those in the investigation of reconstruction schemes on body-fitted unstructured meshes in ref. [20], and we note that the absolute magnitude of error in the present scheme is more than a factor of two lower than was reported by the best scheme in that investigation. The slight tailing-off of the results for mesh 5 is likely the result of using an under-resolved discrete

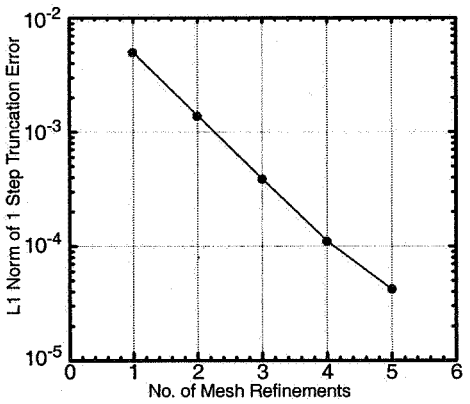


Figure 3: L2 norm of density truncation error for sequence of refined meshes shown in fig. 2.

representation of the geometry, but this issue is still under investigation.

The results shown in fig. 3 were generated using the Colella flux function, without flux limiters. Results with the van Leer option are essentially identical.

Next, we present a validation study examining the flow over an ONERA M6 wing which has been widely cited in the literature. This transonic $M_\infty = 0.84$, $\alpha = 3.06°$ case is often used in the validation of inviscid techniques. This experiment was performed at a relatively high Reynolds number (based on root chord) of 12 x 10^6[21], which minimizes effects of the displacement thickness making accurate comparisons of sectional pressure distributions possible. Other viscous effects in the experimental data are limited to a slight separation which fills in the C_p distribution behind the lambda shock on the lee surface.

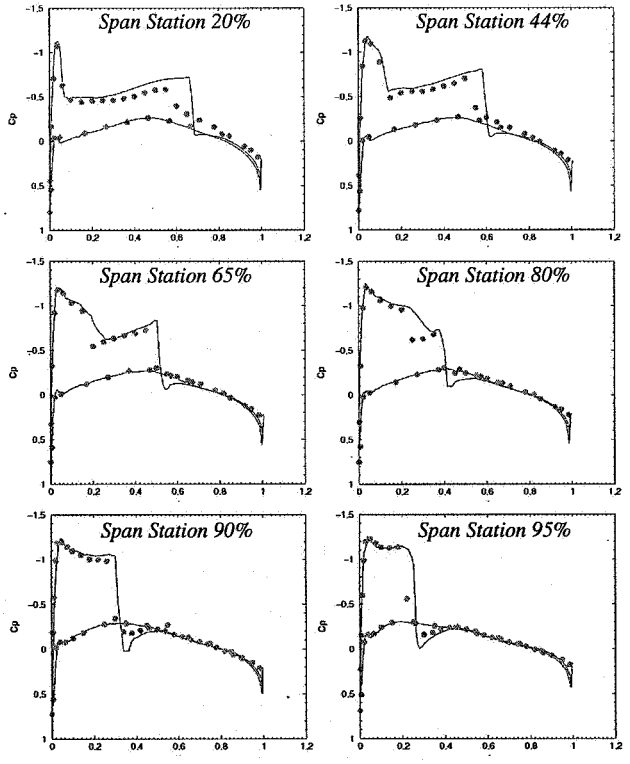


Figure 4: C_p vs. *x/c* for ONERA M6 wing example at six spanwise locations. $M_\infty = 0.84$, $\alpha = 3.06°$. Experimental data from ref. [21] shown as symbols, inviscid discrete solution shown with solid line.

Simulation of this test was conducted using the geometry of a wing in free air, with the far-field boundary located 30 chords from the wing. The final mesh contained 525000 control volumes, with 25000 cut-cells and 528 split-cells. The discrete solution was converged 6 orders of magnitude (in the L1 norm of density) using the van Leer flux function.

Figure 4 provides a quantitative assessment of the solution through pressure profiles at six spanwise stations. This figure displays C_p *vs. x/c* at spanwise stations at 20, 44, 65, 80, 90, and 95% span. The inboard stations correctly display the double-shock on the upper surface, while stations at 90 and 95% confirm accurate prediction of the merging of these shocks. The experimental data at stations 20 and 44% indicate that the rear shock is followed by a mild separation bubble triggered by the shock-boundary layer interaction. As is typical in such cases, the inviscid discrete solution locates this rear shock slightly behind its experimental counterpart.

3 Coarse Mesh Generation and Multigrid

The major decisions one faces in implementing a multigrid scheme concern both the choice of operators for prolongation and restriction and the design of an effective grid coarsening strategy. The development in §3.1 intends to illustrate implementational details concerning the operators used in our approach. §3.2 outlines an automatic coarse grid generation algorithm intended to both adequately represent the solution on the coarse meshes and yet still coarsen fast enough to reduce the computational work.

3.1 Multigrid

The Full Approximation Storage (FAS) scheme we describe is based upon work by Jameson[22]. It is driven by a modified Runge-Kutta smoother which takes the current estimate of the solution, $\bar{q}$, from pseudo–time level n to $n + 1$. At convergence on a grid with spacing h, the smoother produces the discrete solution q_h. The operator L_h represents the spatial discretization on the right-hand-side of eq. (2).

$$L_h q_h = f_h \tag{5}$$

Before convergence, $\bar{q}_h$ does not exactly satisfy the governing equations, producing the fine grid residual r_h.

$$L_h \bar{q}_h - f_h = r_h \tag{6}$$

or

$$L_h \bar{q}_h - L_h q_h = r_h \tag{7}$$

$\bar{q}_h$ differs from the discrete solution q_h by an error term e_h.

$$e_h = \bar{q}_h - q_h \tag{8}$$

Since the high frequency components of e_h can be rapidly annihilated by the fine grid smoother, multigrid seeks to compute the remaining smooth error, e_H, on coarser grids.

$$e_H = \bar{q}_H - \tilde{I}_h^H \bar{q}_h \tag{9}$$

Since FAS is designed for non-linear problems, the coarse grid operator must be applied to both terms on the right-hand-side of eq. (9) rather than to the correction itself. If the smoother has eliminated the higher frequency modes this will be balanced by the restricted residual from the fine mesh.

$$L_H \bar{q}_H - L_H \tilde{I}_h^H \bar{q}_h = -I_h^H r_h \tag{10}$$

In our cell-centered implementation, a volume weighted restriction operator $\tilde{I}_h^H$ transfers the fine solution estimate to the coarse mesh. The restricted residual in a coarse cell (RHS of eq. (10)) is the arithmetic sum of the residuals in the fine grid cells which it encloses.

To ensure that evolution on the coarse grid is driven by the fine grid residuals, the coarse grid forcing function is defined in the usual way.

$$F_H = L_H \tilde{I}_h^H \bar{q}_h - I_h^H r_h \tag{11}$$

Thus, when the fine grid has reached steady state, the fine grid residual vanishes and the coarse grid does not produce any further correction.

After smoothing (or solving) on the coarse grid, the coarse grid correction, $(\bar{q}_H^{n+1} - \tilde{I}_h^H \bar{q}_h^n)$ is brought back to the fine grid state vector through the prolongation operator I_H^h.

$$\bar{q}_h^{n+1} = \bar{q}_h^n + I_H^h (\bar{q}_H^{n+1} - \tilde{I}_h^H \bar{q}_h^n) \tag{12}$$

The preliminary results in this report use direct injection (*i.e.* piecewise constant prolongation). It is well known that such a crude operator introduces high-frequency modes into the fine grid which can adversely affect the overall convergence rate, and generally smoother prolongations are preferred. We have not yet compared direct injection with a linearly interpolated prolongation scheme. The best practical combination of prolongation operators and various multistage schemes[23] will be the topic of future study.

3.2 Automatic Generation of Coarse Grids

A central issue in the implementation of multigrid smoothers on unstructured meshes is the construction of a series of coarse grids. A variety of approaches have been suggested in the literature, however, the asymptotic coarsening ratio in some of these has been insufficient to ensure that the method will extract the full benefit of multigrid. Multigrid does not appear to be widely used in a production setting, and at least part of the reason for this stems from the lack of a robust and automatic mesh coarsening scheme. Additionally, the approach in refs. [2] and [17] permits the cells to divide anisotropically and therefore, we revisit the issue of efficient coarse mesh generation.

In contrast to coarse grid generation problems on general unstructured meshes, cells in a Cartesian mesh are naturally nested. In addition, the cells can be organized such that any cell in the mesh may be uniquely located by a set of integer

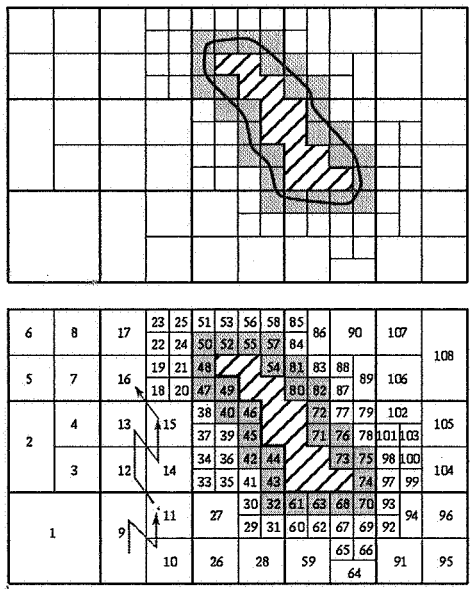


Figure 5: Top: A two dimensional adaptively refined Cartesian mesh. Cut-cells are shown shaded. Bottom: The same mesh, after reordering with a specially designed comparison operator in preparation for coarsening.

indices[2]. The combination of these two facts lead to a novel coarse mesh generation algorithm for adaptively refined Cartesian meshes. The asymptotic complexity of this algorithm is $O(N \log N)$, where N is the number of cells in the fine mesh. This result stems from the fact that a central component of the algorithm is a standard quicksort routine, and all other operations may be performed in linear time.

Our multigrid strategy is based upon the use of a global grid at the finest level consisting of cells at many levels of refinement. This is in contrast to block structured AMR for example, where the finest level grids are not global, but cover only patches of fine cells throughout the domain. Given this global grid, coarser meshes are generated by coalescing the neighboring fine cells into their coarser parent. These fine cells are *siblings* and can coalesce into a single parent as long as they are at the same refinement level. As is standard, cells coarsen by a factor of two (per direction) with each coarsening pass. If one or more in a set of siblings has children of its own, then coarsening is blocked (or "suspended) until those children have been coarsened away. Coarsening is accomplished by the "sibling sort" as described below, and coalescing sets of siblings which are at the same refinement level.

Figure 5 displays a two dimensional, directionally refined Cartesian mesh which illustrates the coarse mesh generation strategy. The mesh shown (top) is the input or "fine mesh" which the algorithm coarsens. The boundary of a hypothetical body is indicated, and the crosshatching indicates where there are no cells in the mesh. Gray shaded cells denote cut-hexahedra. In the lower frame of this figure lies a second view of the mesh after it has been processed by the sibling sort. The cell indices in this mesh indicate the sorted order, which is further illustrated by a partial sketch of the path shown through cells 9–16.

The comparison operator of the sibling sort performs a recursive lexicographical ordering of cells which can coarsen into the same coarse cell. Adaptively refined Cartesian meshes are formed by repeated subdivision of an initial coarse mesh (referred to as the *level 0 mesh*), therefore any cell, i, is traceable to an initial "parent cell" in the level 0 mesh. Similarly, if cell i has been refined R times, it will have parent cells at levels 0 through $(R-1)$. If a cell has never been divided, then it is referred to as a "level 0 cell" and is identical to its level 0 parent.

The sibling sort consists of two steps.

1. Cells on the level 0 mesh are sorted in lexicographic order using the integer coordinates of their level 0 parents as keys.
2. If a cell has been subdivided, recursively sort its children lexicographically.

This algorithm can be implemented with a single quicksort using a comparison operator which examines the integer indices of two input cells on a bit-by-bit basis. As noted above, its asymptotic complexity is proportional to that of the sorting method used.

After sorting the fine mesh, coarsening proceeds in a straightforward manner. Cells are processed by a single sweep through the sorted order. If a contiguous set of cells are found which coarsen to the same parent they are coalesced into that parent. Cells which do not meet this criteria are "not coarsenable" and are injected to the coarse mesh without modification.

The upper frame in Figure 6 shows the coarse mesh resulting from one application of the coarsening algorithm. Notice that cells 16 and 17 in Figure 5 do not coarsen in the first pass since some of their siblings have children (cells 18 through 25). Notice also that fine grid cells on the level-0 mesh are

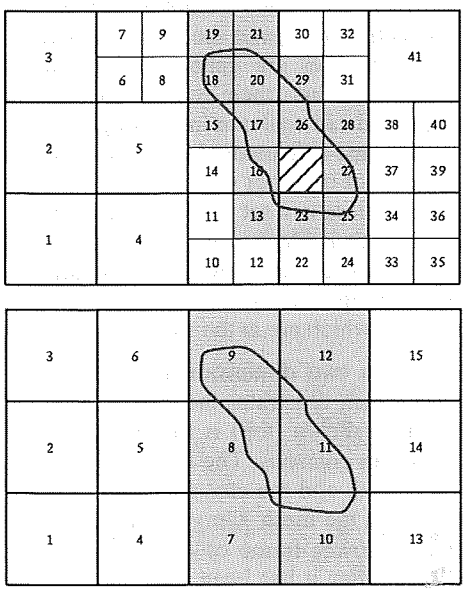


Figure 6: Top: Adapted Cartesian mesh from Figure 5 after one coarsening. Outline of geometry is indicated, and cut-cells are shown in grey. Bottom: Same mesh after one additional application of the coarsening algorithm

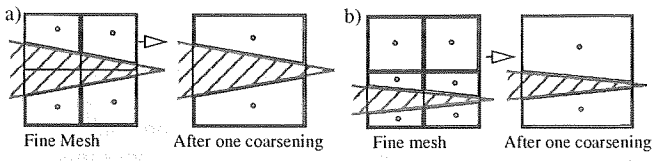


Figure 7: Cell coarsening with split-cells. (a) Four cut-cells become 2 split-cells when the mesh is coarsened, (b) 2 volume cells, and 4 split cells become 2 split-cells after coarsening.

fully coarsened and do not coarsen beyond their initial size (cell 1 for example). The lower frame in Figure 6 shows the mesh resulting from a second application of the coarsening operator. In this second pass, we observe that the coarsened cells in the upper frame were automatically constructed in the sorted order so that this further coarsening does not require any additional sorting.

To summarize, fine grid cells will not coarsen if (1) they are fully coarsened; or (2) one (or more) of a cell's siblings are subdivided. A final restriction is that cells cannot coarsen if doing so will create a difference of more than one refinement level between adjacent cells in the coarsened mesh.

One subtlety that the coarsening algorithm must contend with is indicated in Figure 7. Split-cells on the fine mesh may become merely cut (but not split) cells on the coarse mesh. Conversely, cut-cells on the finest mesh may become split-cells on the coarse mesh. While open questions still exist concerning the level of geometric fidelity required on coarser meshes, we insist that regions of the domain that are disjoint on the fine mesh must remain disjoint in the coarse mesh. The algorithm checks that sibling cells on the fine grid share at least one common face in order to coalesce into the same parent cell on the coarse grid.

Since the work bound of a multigrid scheme depends upon the achievable coarsening ratio, we investigate this ratio on realistic domains in three dimensions. The mesh generator of ref. [2] was used to produce geometry refined meshes for both an ONERA M6 wing and a twin-engine business jet. These meshes varied in size from 7600 to 3.6 M cells. Both were produced from initial background meshes of around 1000 cells. Figure 8 plots the coarsening ratio for these

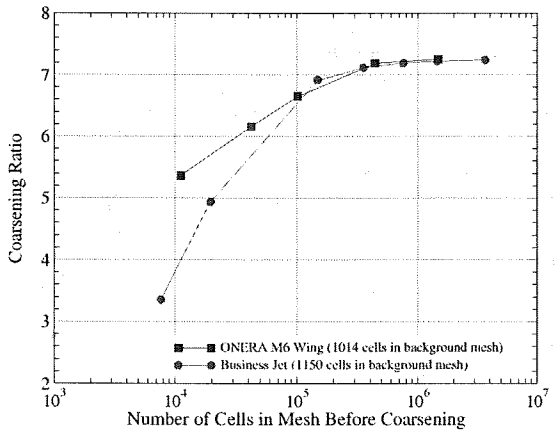


Figure 8: Achievable mesh coarsening ratios on realistic domains in three dimensions.

examples as a function of input mesh size. For both the simple wing, and full aircraft, the coarsening ratio starts out rather low when the meshes are very coarse, but rapidly increases with input mesh size. Both configurations demonstrate mesh coarsening ratios above 4 when the initial mesh contains approximately 10^4 cells, and the coarsening ratio rapidly approaches a limit of around 7.25 at finer resolutions. These results are characteristic of the performance of the coarsening scheme on all examples tested to date.

The four frames in Figure 9 displays a coarsening sequence for the twin engine business jet beginning with the 3.6 M cell mesh from the study in fig. 8. This mesh was successively coarsened to produce the four meshes displayed. The coarsening algorithm achieved ratios of 7.24, 7.16 and 6.49 to produce the coarsest mesh shown (lower right frame) with 10700 cells. With these ratios, the computational effort for a 4-grid multigrid W-cycle would be only 1.38x more than that of a single fine mesh timestep.

3.3 Multigrid Convergence Studies

With the characteristics of the mesh coarsening algorithm established, we begin a very preliminary study of the convergence rate of the multigrid scheme. The transonic ONERA M6 wing case from Figure 4 forms the basis of this initial investigation. The mesh in that computation contained 525000 cells and now forms the finest mesh in the multigrid sequence. Successive coarsening of this mesh by ratios of 7.02, 6.04, and 5.13 produced the four grid sequence used in this study. To prevent limiter fluxuations from contaminating the convergence study, the code was run first-order in this example. Sawtooth W-cycle multigrid was used, with the "optimal 5-stage scheme" of ref. [23] providing time advance.

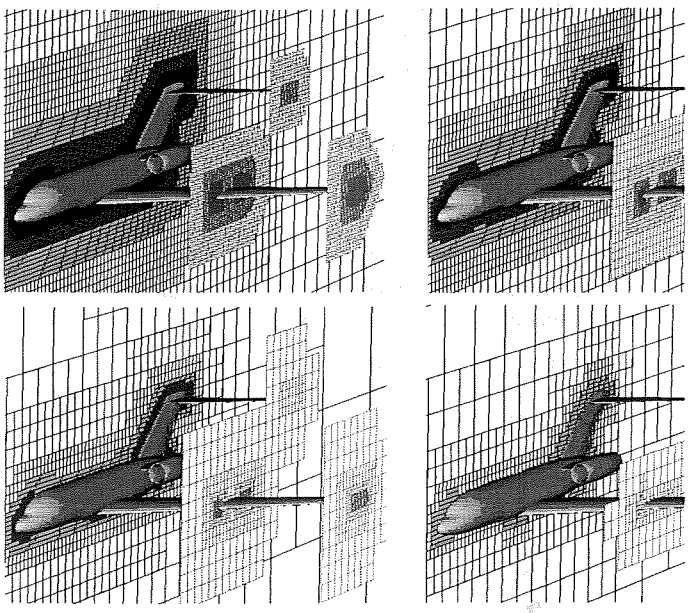


Figure 9: The initial fine (upper left) and first three coarse meshes for a twin engine business jet configuration. The initial mesh has 3.6 x 10^6. Successive coarsening ratios of 7.24, 7.16 and 6.49 produce the coarsest mesh with 10700 cells.

Figure 10 displays convergence behavior both in terms of multigrid cycles and as a function of computational work. The "work" abcissa in the lower figure is normalized by the cost of a single fine mesh residual computation. On 4 meshes, the convergence rate for the first 6 orders of the L1 residual is approximately 0.84. Unfortunatly, this rate decreases over the later iterations to give an overall rate of 0.94 for the entire computation. Despite the slowdown, the lower frame in Figure 5 demonstrates a 5x savings in the computational effort to bring the residual to machine zero.

One likley cause for the slowdown in convergence rate observed in fig. 10 is the formation of shocks in the flowfield over the wing. To investigate the convergence behavior in a shock-free flowfield, the free stream Mach number was reduced to 0.54, leaving all other parameters unchanged. With a fully subsonic flow, flux limiters were turned off and the van Leer scheme was run second-order.

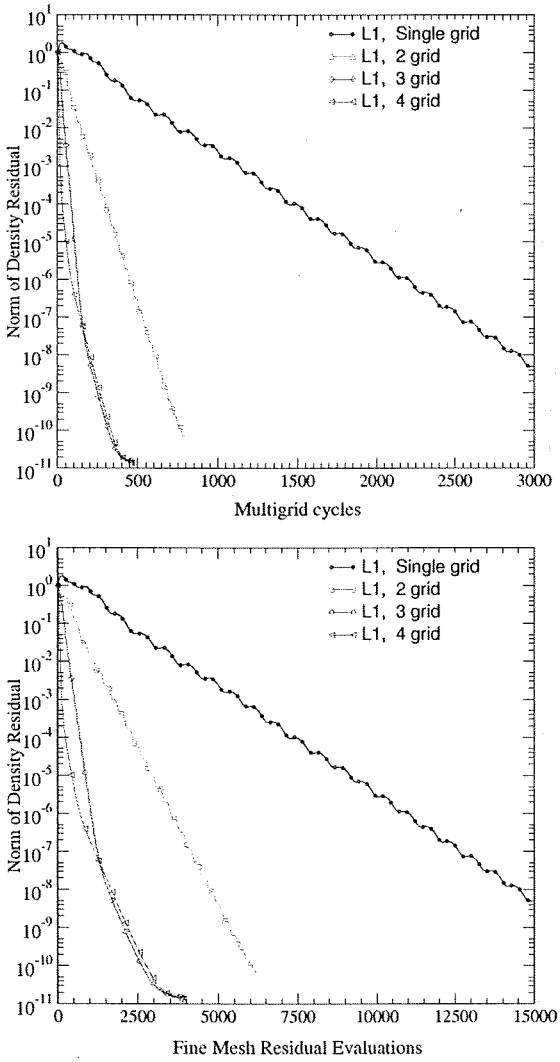


Figure 10: Convergence of transonic ONERA M6 wing test case as a function of both multigrid cycles (upper) and computational work (lower). $M_\infty = 0.84$, $\alpha = 3.06°$.

Figure 11 shows convergence results for this example. Results with 4 and 5 grids both converge at an overall rate of 0.86. While slowdown in convergence observed in the transonic case is much less prominent it still exists. The most obvious culprit is certainly the use of direct injection for prolongation, and future efforts will examine smoother operators and their associated computational cost. Despite this shortcomming, these preliminary results are encouraging and performance is comparable to examples in the literature.

4 Domain Decomposition and Scalability

4.1 Domain decomposition

One novel aspect of this work lies in its approach toward domain decomposition. The option exists to apply a commercial grade uni-processor partitioner like the multi-level nested dissection tool in reference [24] or its multi-processor variant[25]. However, an attractive alternative stems from exploiting the nature of Cartesian meshes. We have built-in a

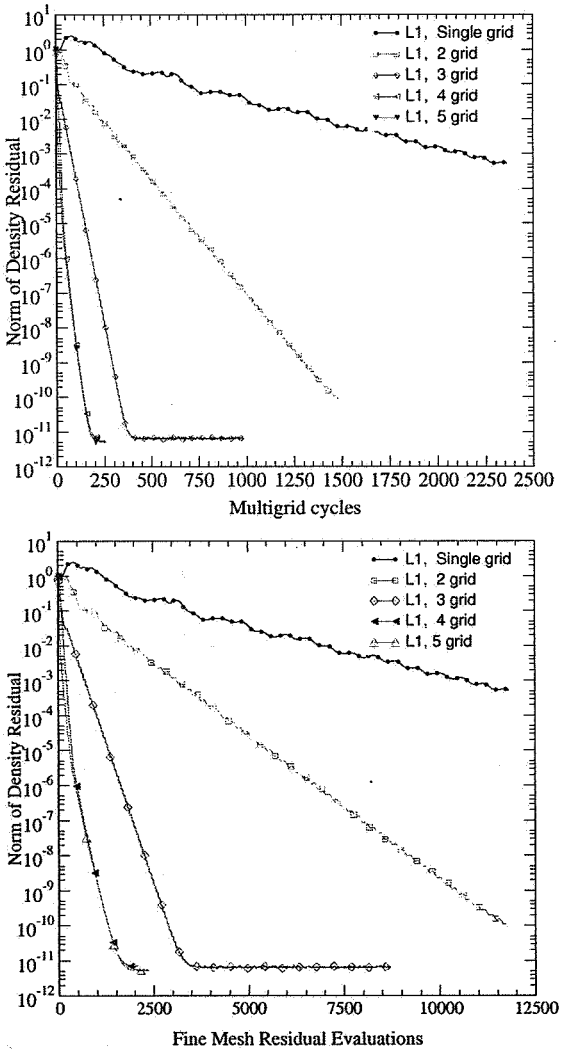


Figure 11: Convergence of subsonic ONERA M6 wing test case as a function of both multigrid cycles (upper) and computational work (lower). $M_\infty = 0.54$, $\alpha = 3.06^\circ$.

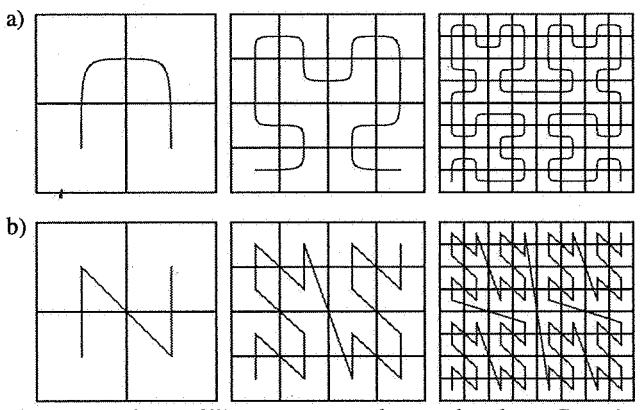


Figure 12: Space-filling curves used to order three Cartesian meshes in two spatial dimensions: a) Peano-Hilbert or "U-ordering", b) Morton or "N-ordering".

partitioner based upon the use of space-filling curves, constructed using either the Morton or Peano-Hilbert orderings[26]. Both of these orderings have been used for the parallel solution of N-body problems in computational physics[27], and the later scheme has been proposed for application to algebraic multigrid[28] in the solution of elliptic PDEs and dynamic repartitioning of adaptive methods[29]. Figure 12 shows both Peano-Hilbert and Morton space-filling curves constructed on Cartesian meshes at three levels of refinement. In two dimensions, the basic building block of the Hilbert curves is a "U" shaped line which visits each of 4 cells in a 2 x 2 block. Each subsequent level divides the previous level's cells by nested dissection, creating subquadrants which are, themselves, visited by U shaped curves as well. This "U-ordering" has locality properties which make it attractive as a partitioner[29]. Similar properties exist for the Morton ordering which uses an "N" shaped curve as its basic building block. Properties and construction rules for these space-filling curves are discussed in refs. [30] and [31]. For the present, we note only that such orderings have 3 important properties.

1. **Mapping** $\Re^d \rightarrow \boldsymbol{U}$: The U and N orderings provide a unique mappings from the d-dimensional physical space of the problem domain $\Re^d$ to a one-dimensional hyperspace, $\boldsymbol{U}$, which one traverses following the curve. In the U-order, two cells adjacent on the curve remain neighbors in this one-dimensional hyperspace.
2. **Locality**: In the U-order, each cell visited by the curve is directly connected to two face-neighboring cells which remain face-neighbors in the one dimensional hyperspace spanned by the curve. Locality in N-ordered domains is almost as good[26].
3. **Compactness**: Encoding and decoding the Hilbert or Morton order requires only local information. Following the integer indexing for Cartesian meshes outlined in ref. [2], a cell's 1-D index in $\boldsymbol{U}$ may be constructed using only that cell's integer coordinates in $\Re^d$ and the maximum number of refinements that exist in the mesh. This aspect is in marked contrast to other parti-

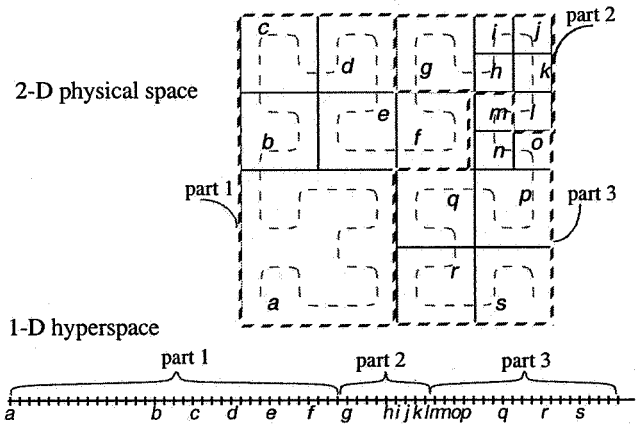


Figure 13: An adapted Cartesian mesh and associated space-filling curve based on the U-ordering of $\Re^2 \rightarrow \boldsymbol{U}$ with the U-ordering illustrating locality and mesh partitioing in two spatial dimensions. Partitions are indicated by the heavy dashed lines in the sketch

tioning schemes based on recursive spectral bisection or other multilevel decomposition approaches which require the entire connectivity matrix of the mesh in order to perform the partitioning.

To illustrate the property of *compactness*, consider the position of a cell i in the N-order. One way to construct this mapping would be from a global operation such as a recursive lexicographic ordering of all cells in the domain. Such a construction would not satisfy the property of *compactness.* Instead, the position of i in the N-order may be deduced solely by inspection of cell i's integer coordinates (x_i, y_i, z_i).

Assume $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$ is the bitwise representation of the integer coordinates (x_i, y_i, z_i) using m-bit integers. The bit sequence $\{\tilde{x}_i^1 \tilde{y}_i^1 \tilde{z}_i^1\}$ denotes a 3-bit integer constructed by interleaving the first bit of x_i, y_i and z_i. One can then immediately compute cell i's position in $\boldsymbol{U}$ as the $3m$-bit integer $\{\tilde{x}_i^1 \tilde{y}_i^1 \tilde{z}_i^1 \tilde{x}_i^2 \tilde{y}_i^2 \tilde{z}_i^2 \ldots \tilde{x}_i^m \tilde{y}_i^m \tilde{z}_i^m\}$. Thus, simply by inspection of a cell's integer coordinates, we are able to directly calculate its position in the one-dimensional space $\boldsymbol{U}$ without any additional information. Similarly compact construction rules exist for the U-order[31].

Figure 13 illustrates these mapping and locality properties for an adapted two-dimensional Cartesian mesh, partitioned into three subdomains. The figure demonstrates the fact that for adapted Cartesian meshes, the hyperspace $\boldsymbol{U}$ may not be fully populated by cells in the mesh. However, since cell indices in $\boldsymbol{U}$ may be explicitly formed, this poses no shortcoming.

The quality of the partitioning resulting from U-ordered meshes have been examined in ref. [29]. and were found to be competitive with respect to other popular partitioners. Weights can be assigned on a cell-by-cell basis. One advantage of using this partitioning strategy stems from the observation that mesh refinement or coarsening simply increases or decreases the population of $\boldsymbol{U}$ while leaving the relative order of elements away from the adaptation unchanged. Remapping the new mesh into new subdomains therefore only moves data at partition boundaries and avoids global remappings when cells adaptively refine during mesh adaptation. Recent experience with a variety of global repartitioners suggest that the communication required to conduct this remapping can be an order of magnitude more expensive than the repartitioning itself[32]. Additionally, since the partitioning is basically just a re-ordering of the mesh cells into the U-order, the entire mesh may be stored as a single domain. At runtime, this single mesh may then be partitioned into any number of subdomains on-the-fly as it is read into the flow solver from mass storage. In a heterogeneous computing environment where the number of available processors may not be known at the time of job submission, the value of such flexibility is self-evident.

The SFC reordering pays additional dividends in cache-performance. The locality property of the SFC produces a connectivity matrix which is tightly clustered regardless of the number of subdomains. Our numerical experiments suggest that SFC reordered meshes have about the same memory cache hit-rate as those reordered using RCM.

Figure 14 shows an example of a three dimensional Cartesian mesh around a triple teardrop configuration partitioned using the U-order. The mesh in this figure contains 24000 cells and is indicated by several cutting planes which have been passed through the mesh, with cells colored by partition number. The upper frame shows the mesh and partition boundaries, while the lower frame offers further detail through an exploded view of the same mesh.

In determining partition boundaries in this particular example, cut-cells were weighted 10x as compared to un-cut Car-

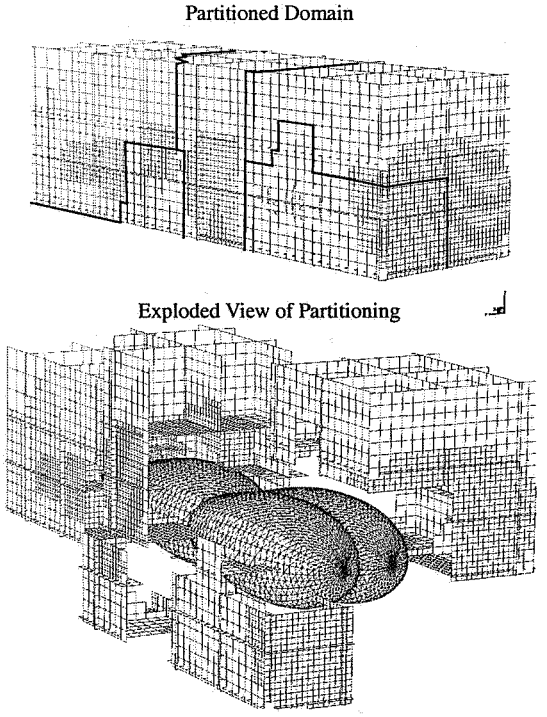


Figure 14: Partitioning of 6 level adapted mesh around a tripl teardrop geometry with 24000 cells into four subdomain using space-filling curves. The mesh is shown by a collectio of cutting planes through each partition.

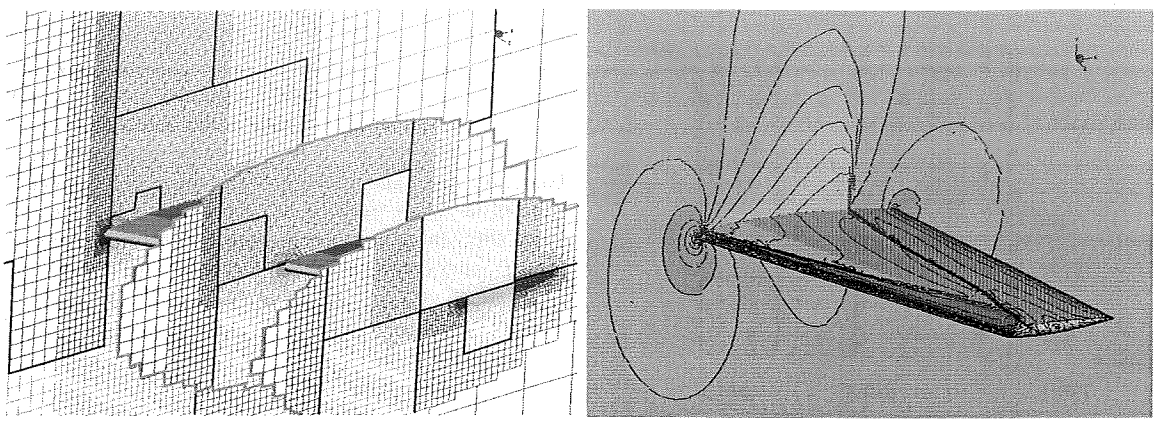


Figure 15: Partitioned mesh and C_p contours for the ONERA M6 wing example. The mesh contains 525000 cells at 9 levels of refinement, mesh partitions are shown by color-coding and outlined in heavy lines. C_p contours are plotted using a cell-by-cell reconstruction of the discrete solution. $M_\infty = 0.84$, $\alpha = 3.06$, van Leer flux.

tesian hexahedra. Numerical experiments indicate that a wieghting foactor of 2.7 provides better load-balancing, and this factor was used in all the results presented in the following section.

4.2 Scalability and Performance

Scalability tests were conducted on the 525000 cell transonic ONERA M6 wing example from fig. 4 (§2.3) using from 1 to 32 processors on a 250Mhz Mips R10000 based SGI Origin 2000. Figure 15 shows a view of the partitioned mesh (8 subdomains) accompanied by a contour plot of pressure coefficient in the discrete solution. Each processor of this platform has a 4Mb Level 2 cache, and two processors on each board share the same local memory. Examination of this plot shows generally good scalability, however, communication does appear to slow this particular computation on 4 and 8 processors when the problem initially gets spread over several boards within the machine. On 32 processors the timings show a "cache bubble" evidenced by the fact that the results on 32 processors are more than a factor of two faster than the timings on 16 processors. Table 1 shows the per-processor execution rate and parallel speed-up for this example. Results in this table show a 4% increase in per-processor execution rate on 32 processors as each processor's L2 cache was very nearly sufficient to store the individual subdomains. The table demonstrates no substantial decrease in performance with larger numbers of subdomains. Results in Table 1 and in Figure 16 were obtained by averaging the results of 3 separate sets of tests since timings on this machine are known to vary by as much as 10%.

Table 1: Parallel scalability and processing rate per processor. Results for each partitioning reflect average of three runs. ONERA M6 wing, 525000 control volumes, 200 iterations per test.

No. of Processors	CPU time/ CPU (sec.)	Parallel Speed-up	Mflops/ CPU[a]	Ideal Speedup
1	2559	1	81.4	1
2	1315	1.94	81.9	2
4	865	2.96	72.77	4
8	383	5.76	77.57	8
16	188	13.61	78	16
32	90	28.37	82	32

a. MFLOPS counted using 250MHz R10000 hardware counters on optimized code, with single cycle MADD instruction disabled. Floating-point multiply, add, and divide each counted as one flop.

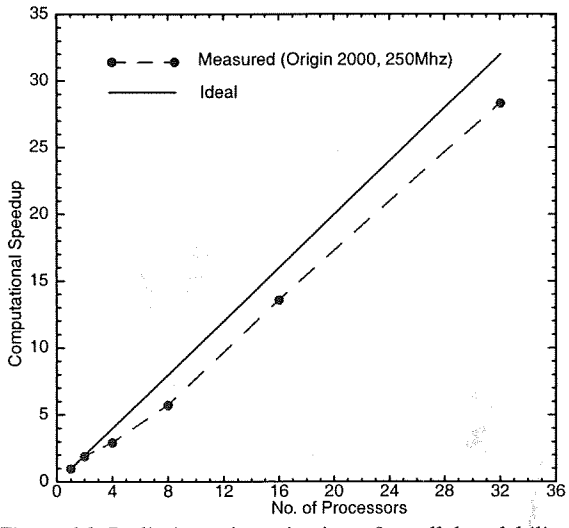


Figure 16: Preliminary investigation of parallel scalability of single mesh (no-multigrid) ONERA M6 wing case. Data reflect average results from 3 runs with each partitioning.

Figures 17 and 18 provide a final look at solver scalability. This example shows the twin engine business jet from fig. 9 run at a freestream Mach number of 0.84 and 1.81° angle of attack. Figure 17 shows a contour plot of pressure coefficient on the body surface and in the symmetry plane. In this example the mesh contained 1.42 M cells at 11 levels of refinement. The case was run on from 1 to 64 processors on the same machine as the ONERA M6 example. Figure 18 shows a plot of computational speedup against number of CPU's. Performance is comparable to that of the preceeding case, and on 64 processors, this example achieved a computational speedup of 52.3.

5 Conclusions and Future Work

This paper presented preliminary verification and validation of a new, parallel, multilevel Euler solver for Cartesian

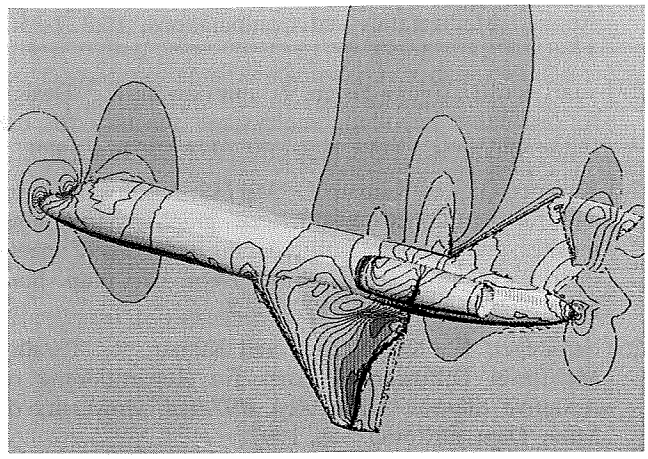


Figure 17: Symmetry plane and surface pressure coefficient for twin engine business jet computed on 64 processors for scalability testing. The mesh contains 1.42 M cells at 11 levels of refinement. *Cp* contours are plotted using cell-by-cell reconstruction of the discrete solution, $M_\infty = 0.84$, $\alpha = 1.81°$, van Leer flux

meshes. Comparison of the scheme's local truncation error with an analytic solution demonstrated an achieved order of accuracy between 1.82 and 1.88. Preliminary validation by direct comparison to experimental results on a three dimensional wing configuration was also performed, demonstrating that the discrete solutions were competitive with other solution procedures.

Documentation of a new on-the-fly SFC based parallel domain decomposition strategy was also presented. This strategy enables SFC reordered meshes to be pre-sorted and stored as a single domain. This mesh can then be decomposed into any number of partitions in at run time. Investigations demonstrated that this decomposition strategy produces a parallel speed-up in excess of 52 on 64 processors.

Details of a new automatic coarse mesh generation algorithm for multilevel smoothers on adaptively refined Cartesian

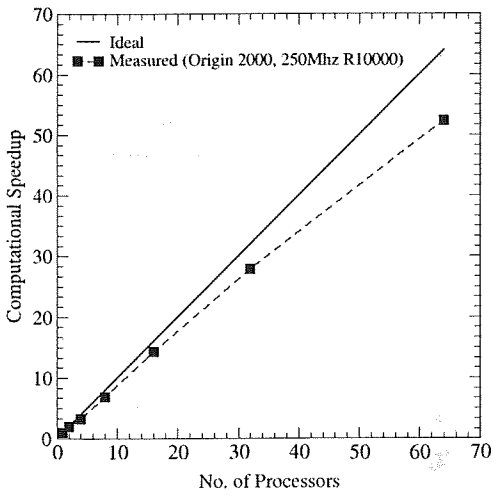


Figure 18: Preliminary investigation of parallel scalability of twin-engine business jet case.

meshes were also presented. In this approach, every mesh in the multigrid hierarchy covers the entire computational domain. Examples on realistically complex three dimensional configurations demonstrated that this algorithm generally achieves mesh coarsening ratios in excess of 7 on adaptively refined meshes. Preliminary convergence results for W-cycle multigrid on simple geometries produced convergence rates between 0.84 and 0.94.

The next step in our development focuses upon scalability of the multigrid method, and an examination of the trade-offs between smoothness of the prolongation operator and computational expense. Future investigations will consider the degree of geometric fidelity required to maintain good asymptotic performance of the multigrid smoother.

Acknowledgements

The authors would like to thank both R. Löhner and S. Baden for pointing to literature on the use of space-filling curves as potentially useful reorderers and partitioners. M. Berger and G. Adomavicius were supported in part by AFOSR Grant F49620-97-1-0322 and DOE Grant DEFG02-92ER25139. Some of this work was performed while M. Berger was at RIACS and this support is gratefully acknowledged.

6 References

[1] Melton J. E., Berger, M. J., and Aftosmis, M. J., "3D Applications of a Cartesian grid Euler method," *AIAA Paper 95-0853-CP*, Jul. 1993

[2] Aftosmis, M.J., Berger, M.J., Melton, J.E.: "Robust and efficient Cartesian mesh generation for component-based geometry." *AIAA Paper 97-0196*, Jan. 1997.

[3] Aftosmis, M.J., Melton, J.E., and Berger, M.J., "Adaptation and Surface Modeling for Cartesian Mesh Methods," AIAA Paper 95-1725-CP, Jun., 1995.

[4] Charlton, E. F., and Powell, K. G., "An octree solution to conservation-laws over arbitrary regions." *AIAA Paper 97-0198*, Jan. 1997.

[5] Wang Z.J., "An automated viscous adaptive Cartesian grid generation method for complex geometries." in *Proceedings of the 6th International Conf. on Numerical Grid Generation in Computational Field Simulations,* Eds. Cross, M. *et al.,* Univ. Greenwich, UK., 1998.

[6] Day, M. S., Colella, P., Lijewski, M. J., Rendleman, C. A., and Marcus, D. L., "Embedded boundary algorithms for solving the Poisson equation on complex domains," Lawrence Berkeley National Laboratory, *LBNL-41811*, May, 1998.

[7] Karman, S. L., "SPLITFLOW: A 3D unstructured Cartesian/prismatic grid CFD code for complex geometries." *AIAA Paper 95-0343*, Jan. 1995.

[8] Forrer, H., "Second order accurate boundary treatment for Cartesian grid methods." Seminar for Angewandte Mathmatic, ETH Zürich, ETH Research Report 96-13, 1996.

[9] Melton, J.E., *Automated Three-Dimensional Cartesian Grid Generation and Euler Flow Solutions for Arbitrary Geometries*, Ph.D. thesis, Univ. CA. Davis CA, 1996.

[10] Berger, M.J and Melton. J.E., "An accuracy test of a Cartesian grid method for steady flows in complex geometries." *Proc. 5th Intl. Conf. Hyperbolic Prob.*, Stonybrook NY, Jun. 1994. also RIACS report 95-02, NASA Ames, Feb., 1995.

[11] Colella P, Ferguson, R., and Glaz, H., "Multifluid algorithms for Eulerian finite difference methods". Preprint 1996.

[12] Coirier, W.J., "An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler Equations," *NASA TM-106754*, Oct., 1994. also Ph.D. Thesis, Univ. of Mich., Dept. of Aero. and Astro. Engr., 1994.ill's thesis

[13] Griebel, M., Tilman, N., and Regler, H., "Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries." Int. J. Numer. Methods for Heat and Fluid Flow 26, pp. 281-301, 1998, also as SFB report 342/1/96A, Institut für Informatik, TU München, 1996.

[14] De Zeeuw, D., and Powell, K., "An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations," *AIAA Paper 91-1542*, Jun., 1991.

[15] Anderson, W.K., Thomas, J.L., and van Leer, B., "A comparison of finite volume flux vector splittings for the Euler equations" *AIAA Paper 85-0122*, Jan. 1985

[16] Venkatakrishnan, V., "On the accuracy of limiters and vonvergence to steady state solutions." *AIAA Paper 93-0880*, Jan. 1993.

[17] Berger, M.J and Melton. J.E., "An accuracy test of a Cartesian grid method for steady flows in complex geometries." Proc. 5th Intl. Conf. Hyperbolic Prob., Stonybrook NY, Jun. 1994. also RIACS report 95-02, NASA Ames, Feb., 1995.

[18] Kreiss, H.O., Manteuffel, T.A., Swartz, B., Wendroff, B., and White, A.B., "Supraconvergent schemes on irregular grids." *Math Comp.* **47**, 1986.

[19] Wendroff, B., and White, A.B. "A supraconvergent scheme for nonlinear hyperbolic systems." Comput. Math. Appl., **18**, 1989.

[20] Aftosmis, M. J., Gaitonde, D., and Tavares, T. S., "Behavior of linear reconstruction techniques on unstructured meshes." *AIAA J.*, **33**(11), pp. 2038-2049, Nov. 1995.

[21] Schmitt, V., and Charpin, F., "Pressure distributions on the ONERA-M6-Wing at transonic Mach numbers." *Experimental Data Base for Computer Program Assessment*, AGARD Advisory Report AR-138, 1979.

[22] Jameson, A.,"Solution of the Euler equations for two dimensional transonic flow by a multigrid method." Applied Mathematics and Computations, **13**:327-356. 1983.

[23] van Leer, B., Chang-Hsien, T., and Powell, K., "Design fo optimally smoothing multi-stage schemes for the Euler equations." *AIAA Paper 89-1933-CP*, Jun. 1989.

[24] Karypis, G., and Kumar, V., "METIS: A software package for partitioned unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices." University of Minn. Dept. of Comp. Sci., Minneapolis, MN., Nov. 1997

[25] Schloegel, K., Karypis, G., and Kumar, V., "Parallel Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes." *Tech. Rep. #97-014*, University of Minn. Dept. of Comp. Sci., 1997.

[26] Samet, *The design and analysis of spatial data structures*. Addison-Wesley Series on Computer science and information processing, Addison-Wesley Publishing Co., 1990.

[27] Salmon, J.K., Warren, M.S., and Winckelmans, G.S., "Fast parallel tree codes for gravitational and fluid dynamical N-body problems." *Internat. Jol. for Supercomp. Applic.* **8**:(2), 1994.

[28] Griebel, M., Tilman, N., and Regler, H., "Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries." Int. J. Numer. Methods for Heat and Fluid Flow 26, pp. 281-301, 1998, also as SFB report 342/1/96A, Institut für Informatik, TU München, 1996.

[29] Pilkington, J.R., and Baden, S.B.,"Dynamic partitioning of non-uniform structured workloads with spacefilling curves." Jan. 1995.

[30] Schrack, G., and Lu, X., "The spatial U-order and some of its mathematical characteristics." *Proceedings of the IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing.* Victoia B.C, Canada, May, 1995.

[31] Liu, X., and Schrack, G., "Encoding and decoding the Hilbert order." *Software-Practice and Experience*, **26**(12), pp. 1335-1346, Dec. 1996.

[32] Biswas, R., Oliker, L., "Experiments with repartitioning and load balancing adaptive meshes." NAS Technical Report NAS-97-021, NASA Ames Research Ctr., Moffett Field CA., Oct. 1997.